

Consolidation and Review

Weeks 1–5 on one page, the LMS so far, and how to prepare for the midterm

No new lab

Catch-up and revision

Midterm is next week

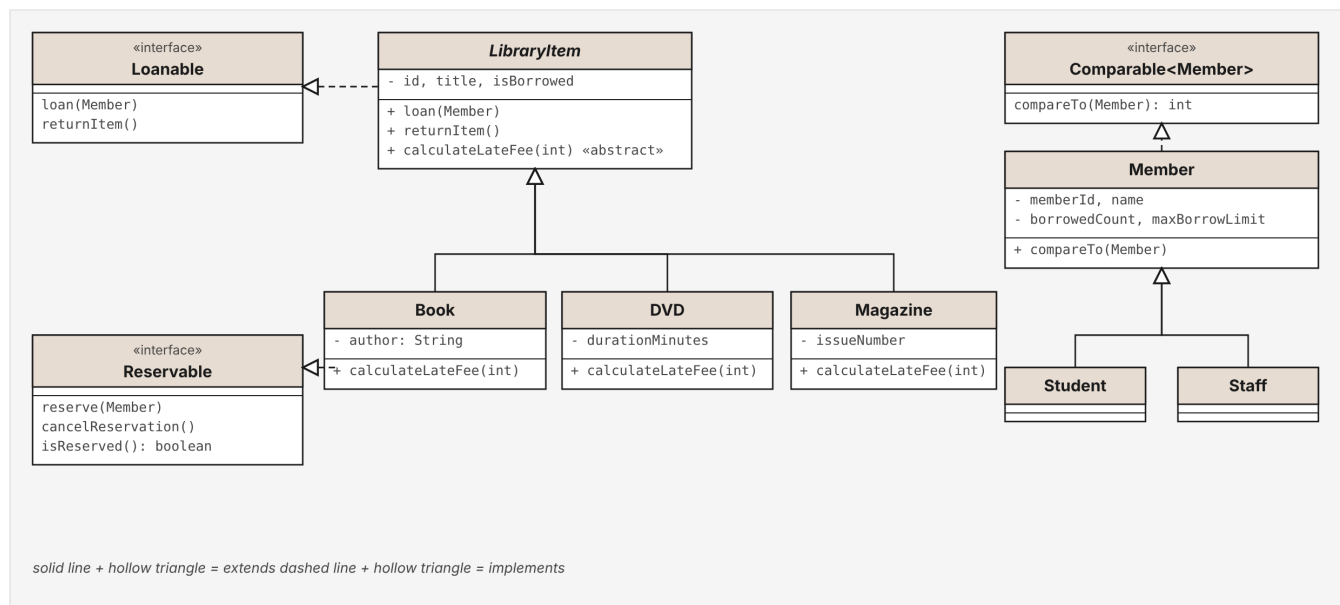
How to use this week

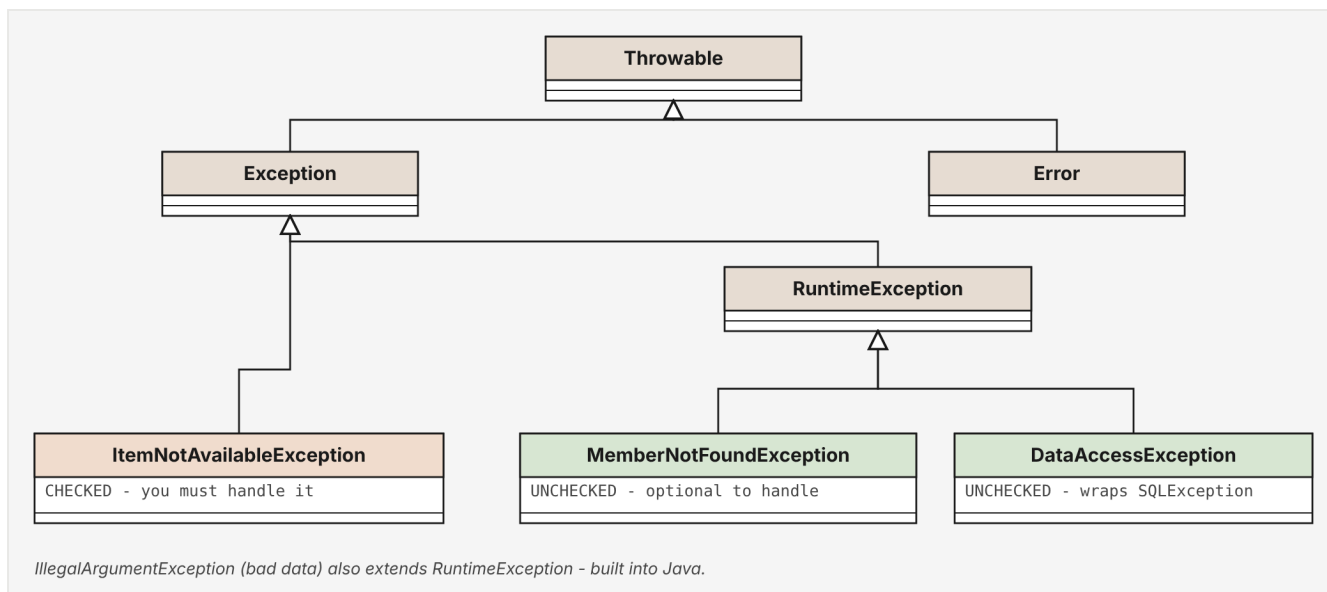
There is **no new lab**. Use the time to (1) finish and merge any unfinished PR, (2) fix review comments, (3) rebuild the LMS **from a blank folder without looking**, and (4) practise with the questions at the end.

1. Five weeks on one page

Week	Big idea	Keywords	LMS artefact
1	Objects bundle state and behaviour	class, object, field, method, constructor, <code>this</code> , UML	<code>Book</code>
2	Hide the data, guard the door	<code>private</code> , getter, setter, validation, composition	<code>Book</code> , <code>Member</code> , <code>Library</code>
3	Reuse and "many forms"	<code>extends</code> , <code>super</code> , <code>@Override</code> , overloading, static vs dynamic binding, <code>instanceof</code>	<code>LibraryItem</code> hierarchy
4	Programme to abilities, not classes	<code>abstract</code> , <code>interface</code> , <code>implements</code> , <code>Comparable</code>	<code>Loanable</code> , <code>Reservable</code> , <code>Magazine</code>
5	Signal errors, let callers decide	<code>try/catch/finally</code> , <code>throw</code> , <code>throws</code> , checked vs unchecked	<code>ItemNotAvailableException</code>

2. The LMS so far





3. Concepts students most often confuse

Confusion	The clear version
Class vs object	Class = blueprint. Object = one thing built from it with <code>new</code> .
<code>private</code> means "secret"	It means "only this class's code". Other classes use public methods.
Overriding vs overloading	Override = same signature in a subclass, chosen at run time . Overload = same name, different parameters, chosen at compile time .
Abstract class vs interface	is-a with shared code vs can-do contract. One parent, many interfaces.
Checked vs unchecked	Checked: compiler forces handling. Unchecked (<code>RuntimeException</code>): optional.
Declared type vs real type	<code>LibraryItem x = new DVD(...)</code> : declared <code>LibraryItem</code> , real <code>DVD</code> . The <i>declared</i> type decides what you may call; the <i>real</i> type decides which overridden version runs.

4. Trace these by hand (no computer!)

```

LibraryItem a = new Book("B1", "Clean Code", "Martin");
LibraryItem b = new DVD("D1", "Algorithms", 120);
System.out.println(a.calculateLateFee(2));
System.out.println(b.calculateLateFee(2));

```

```

try {
    library.borrowItem("B1", "M1");
    System.out.println("A");
    library.borrowItem("B1", "M2");
    System.out.println("B");
} catch (ItemNotAvailableException e) {
    System.out.println("C");
} finally {
    System.out.println("D");
}

```

(Answers: 20.0 and 50.0; then A C D, assuming M1 and M2 exist and B1 is free at the start.)

5. Practice questions

1. Explain, with a library example, how encapsulation protects an object.
2. Draw a UML class diagram for `Member`, `Student`, `Staff` with fields and one method each.
3. A `Magazine` class extends `LibraryItem` but does not compile. Give two possible reasons.
4. Write an interface `Renewable` with one method, and a class that implements it.
5. What is the difference between `throw` and `throws`?
6. Why is this method a poor use of `instanceof`, and how would you fix it using polymorphism?

```
double fee(LibraryItem i, int days) {  
    if (i instanceof Book) return days * 10.0;  
    else if (i instanceof DVD) return days * 25.0;  
    else return days * 5.0;  
}
```

(Hint for 6: it is a type check where each subclass should override `calculateLateFee`.)

6. Rebuild challenge

On a clean branch `week6-rebuild` (do not merge it), recreate: `LibraryItem` (abstract), `Book`, `DVD`, `Member`, `Student`, `Library.borrowItem`, one custom exception, and a `Main` that triggers it. Time yourself. Under 45 minutes with no notes means you are ready.

7. Revision checklist

- ☐ All of Labs 1–4 merged, with UML in `docs/`
- ☐ I can write a class with private fields, validated setters and a constructor from memory
- ☐ I can explain static vs dynamic binding using the `addItem` and `calculateLateFee` examples
- ☐ I can say when I would use an abstract class and when an interface
- ☐ I can write `try/catch/finally` and a custom exception without help